BTeV Trigger/DAQ Innovations

Author(s) Name(s)
Author Affiliation(s)
E-mail

Abstract

The BTeV experiment was a collider based HEP B-physics experiment proposed at Fermilab. It included a large-scale, high speed trigger/data acquisition (DAQ) system, reading data off the detector at 500 Gbytes/sec and writing to mass storage at 200 Mbytes/sec. The online design was considered to be highly credible in terms of technical feasibility, schedule and cost. This paper will give an overview of the overall trigger/DAQ architecture, highlight some of the challenges, and describe the BTeV approach to solving some of the technical challenges.

At the time of termination in early 2005, the experiment had just passed its baseline review with flying colors. Although not fully implemented, many of the architecture choices, design, and prototype work for the online system (both trigger and DAQ) were well on their way to completion. Other large, high-speed online systems may have interest in the some of the design choices and directions of BTeV, including (a) a commodity-based L1 tracking trigger running asynchronously at full rate, (b) the hierarchical control and fault tolerance in a large real time environment, (c) a partitioning model that supports offline processing on the online farms during idle periods with plans for dynamic load balancing, and (d) an independent parallel highway architecture.

1. Introduction

The proposed BTeV detector consisted of 6 separate subdetectors: pixel, silicon strips, straw tubes, rich, emcal and muon with the pixel detector dominating the channel count (see *Table 1*).

Subsystem	Channels	DCB Subsystems
Pixel	21M	10
Strips	128K	2
Straws	54K	
Rich	154K	
EMCAL	10K	
Muon	37K	

Table 1. Channel Count

The overall detector was designed to run with a 7.5 MHz clock rate delivering beam with a 396nsec (2.5 MHz) spacing between. Three levels of triggering reduced the overall acceptance rate to 2.5 KHz. See *Table 2* for the data rates at each trigger leve.

	Frequency	**Event Size**	**Data Rate**
Into L1	2.5 MHz	200 KB	500 GB/sec
Into L2/L3	50 KHz	250 KB	12.5 GB/sec
Into archival	2.5 KHz	80 KB	200 MB/sec

Table 2. Event Rates

Figure 1 shows the overall architecture of the entire BTeV trigger and DAQ.

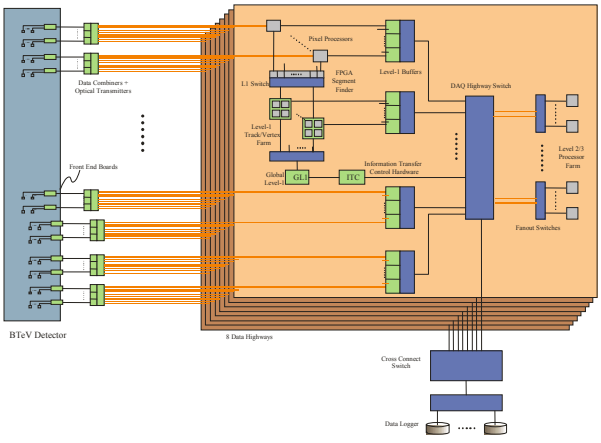

Figure 1. Trigger/DAQ Overview

There are several points Figure 1 worth noting. First, the detector was unique in that all the data was brought off the detector and digitized in subdetector-specific front end boards. This data was sent via point to point copper cable to data combiner boards (DCBs) before being sent to the first level trigger. The data combiner board design was common across subdetectors (FN). This architecture benefited the design in that 1) much of the L1 trigger could be done in software which allowed for the commodity components described in Mike Wangs talk 2) the DCBs provided a single entry point into the trigger/DAQ that could be centrally designed and maintained reducing the long term support load.

Secondly, data collected in the DCBs were routed to 8 independent, parallel highways. See section for a more detailed description of the DCB design. This design reduced the overall control overheads on each particular highway and grouped data coming out of the L1 buffers into larget packets for better Ethernet performance. Thirdly, the level 2 and 3 trigger (L2 and L3, respectively) decisions were made on the same L2/L3 farm. The practical difference is the amount of data from a particular event is being evaluated and the physics algorithms deployed. Lastly, the baseline of the BTeV architecture was to log experiment data to large disk farms, ie no tape, using dCache as the underlying data storage support software.

4. Highway

Let's look at the highway architecture in more detail starting with the DCB.

The data combiner boards are a custom component and the first interface between the subdetectors' front end electronics and Two flavors of this board exist – one for the fpix board readout (Pixel and Strip detectors) and one for everyone else. For the purposes of this paper, they can be considered the same with only a variation on the configuration of the input ports and rates.

DCBs are the place in the online architecture that splits the detector data into highways – eight parallel and independent data streams each processing 1/8th of the detector data. The original DCB design routed a crossing a time. This was resulting a complex routing table algorithm to factor out any periodicity in the tevatron. We were investigating the possibility of routing a data a turn at a time. This alternative approach would simplify the routine, but increase the length of time data would live in the DCB exposing it to a higher single even upset rate. A given crossing will send all of its data to only 1 highway.

A DCB board contained a commercial CPU and fast Ethernet interface for control and low speed data readout for diagnostics and commissioning.

Figure 2 shows the data and control flow in/out of a DCB board.

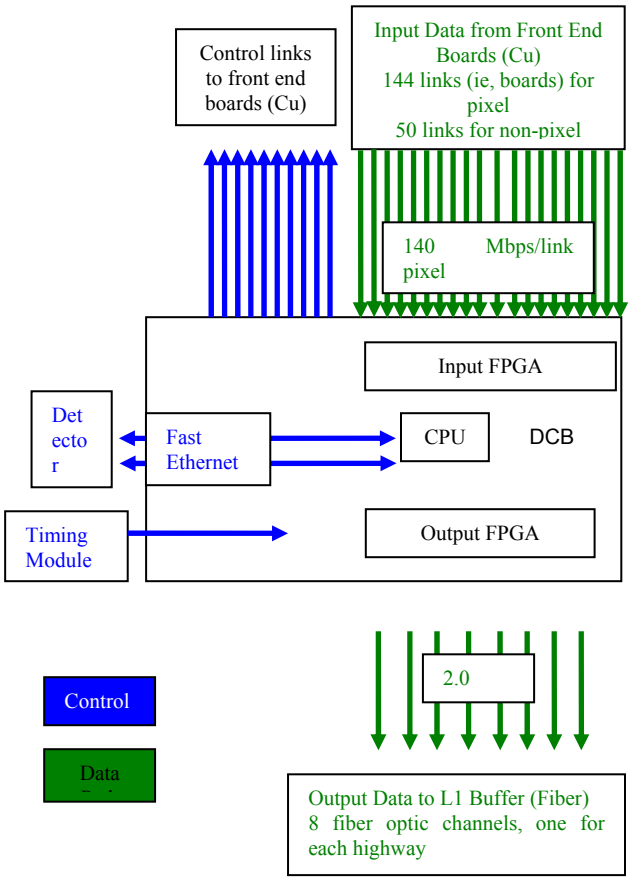

Figure 2: DCB Data/Control Flow

The DCBs are received precise clock information from the timing system We imagined being able to reset/reconfigure "live" on a future crossing. E.g, if a catastrophic highway failure, the DCBs could be sent a control command to the DCBs to route to 7 highways instead of 8 starting at crossing number 6000.

Data from the DCBs were routed over point to point optical links into L1 buffers, another custom electronics component. Each L1 Buffer could communicate with 24 DCBs (two crates worth). See It's this unit that we will talk about in the partitioning section regarding resource reservation.

For the two detectors that fed into the L1 trigger (pixel and muon), data from the DCBs were also read out by the segment preprocessors. Calculated data and

decisions from the L1 trigger would then fed into additional L1 buffers.

The primary responsibility of an L1 buffer is to buffer the detector data for a long enough time to make an L1 trigger decision. There were 24 L1 buffers per highway with 3G of memory each (the smallest we imagined would be available). For an aggregate total of ~0.5 TBytes of memory, or roughly 1 second of data. Its flow diagram in shown in Figure 3.

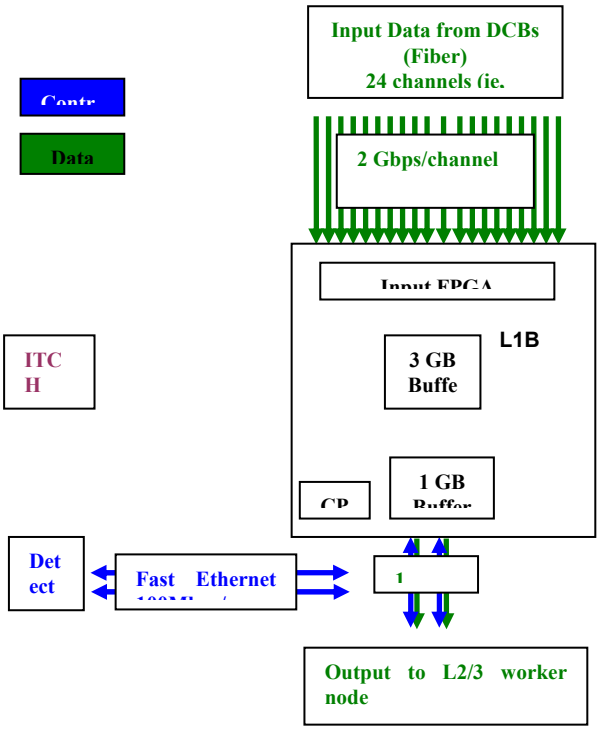

Figure 3. L1 Buffer Data/Control Flow

Data comes into the input FPGA and is stored in a dcircular buffer. When a L1accept is received, all data for that subevent is copied into a smaller 1GB memory area that will be used when transferring accepted events to the downstream L2 trigger farm. No special information is needed from the ITCH regarding L1 rejects; the data is simply overwritten in the circular buffer.

This separate L1 trigger accept buffer in memory will help in distributing an event to a second L2 worker node. This capability might of interest when a specific event satisfied the trigger tables in more than one partition (see section blah). Details need to be flushed out regarding bookkeeping in the L1 buffer to understand exactly when an accepted event can be overwritten (ie, when it has successfully transferred to the primary partition) while allowing it to stay in memory as long as possible to allow for parasitic L2 nodes to transfer additional copies.

From the L1 buffers, data is sent over a GBit Ethernet to the to the L2/L3 trigger farm The L2/L3 trigger farm is really running on the same hardware with the distinction being which physics algorithm would be currently processing the data. It consisted of commodity processors was also split into highways. Data from one highway could be physically sent to another highway, but at a price to performance. Each highway consisted of about 90 worker nodes each with dual CPUs.

To reduce the control overhead and complexity of the software, we designed the event building switch with enough capacity to send a complete event at L2. Each CPU box was referred to as a worker node. Workers nodes would declare themselves to a particular partition, ie, trigger list (see section on partitioning) and notify the ITCH when they were ready for data. The ITCH would assign them a particular crossing number. All data from that crossing would be sent to that worker node.

Worker nodes themselves were grouped into functional units in a highway. Each group was controlled by a regional manager consisiting of 12 worker nodes. A regional manager was responsible for configuring its aassociated worker nodes, fanning out control commands and collecting statuses, caching DMBS data (eg, various versions of the trigger algorithm), and handling regional faults.

3. Fault tolerance

The BTeV trigger performs sophisticated computations using large ensembles of FPGAs, embedded processors, and conventional microprocessors. This system will have between 5,000 and 10,000 computing elements and many networks and data switches. The need for fault-tolerant, fault-adaptive, and flexible techniques and software to manage this huge computing platform has been identified as one of the most challenging aspects of this project.

As a response to this challenge, the Real Time Embedded Systems (RTES) project group was formed and funded through a 5 year NSF grant. This research group is a collaborative effort between computer scientists and high energy physicists. It is researching the design and implementation of high-performance, heterogenous, reliable, and fault-adaptive real-time

systems that are embedded (i.e. are an integral part of the hardware they serve).

4. L1 trigger

One particular highlight of the BTeV design is a commodity based L1 tracking trigger running asynchronously at full rate. Details

5. Partitioning

Partitioning the detector was defined to be running multiple independent data acquisition systems in parallel. The value of partitioning and when would partition the detector are different depending on the phase of the project. I.e., partitioning needs during commissiong e.g., testing the subdetectors in parallel may be different than when testing new L3 trigger algorithms while taking physics quality data. Additionally, the BTeV L2/L3 farm contained A LOT of processing power. It was an expensive investment to be idle during periods of no beam or low luminosity. The experiment was relying on using the online farm to do offline processing when not required for data taking. We planned to use the concept of partitioning to support this functionality by having an "offline" partition.

Partitioning is strictly a logical concept. One must map this onto the physical implementation of the experiment. The online DAQ/trigger was constructed in 2 stages to match the proposed funding profile with roughly 50% of capacity at each stage. The first stage consisted of 4 highways and the second stage added the next fiscal year. Even within a stage, individual highways were commissioned one at a time. The logical concept of partitioning needed to support running multiple partitions on a single highway (when only 1 was constructed) as well as the final system with 8 highways

The parallel highway architecture and dynamic reloading of DCB routing tables gave BTeV overwhelming flexibility in this regard, so much so that it became a source of confusion. It's important to note that at the time of BTeV's cancellation, the detailed requirements and design of partitioning were a very hot topic among the online staff and hadn't yet reached buy in from the collaboration. What we will discuss here are some of the ideas.

First, we imagined the cycle of running a partition involved the following steps:

Selecting/reserving [subset of] electronics to be read out
Defining how much L2/L3 trigger processing power needed
Initializing the hardware
Collecting the data
Freeing the resources

Original ideas were for a physicist to select the strips and straw front end crates, request 50 Mflops of L2/L3 nodes for processing, and then let software map this out onto a physical implementation. Depending on how many nodes might be needed, the layout may be to run on a single highway or to route to n highways. If a given front end crate needed to send data to multiple partitions, its DCB routing tables would be dynamically reset to route as necessary. This idea was balked at by members of the collaboration believing nobody would every have any idea where the events were going. An additional constraint on the system was that, because of it's architecture, the L1 trigger hardware on a particular highway could NOT be partition; however multiple sets of trigger tables could be loaded into Global Level 1. It was imagined that a given L2/L3 node could belong to one and only one partition.

Because of the sheer number of electronics involved in the Trigger/DAQ, we were converging on the idea of the L1 trigger and active highways always being available as a shared resource. A human run coordinator would establish the overall online configuration for a period of time (day/week/month) and coordinate the individual groups taking data during this period. This stable configuration period had a fixed and predefined set of allowable highways. Subdetector groups still had to select the specific electronics to read out (in units of L1 buffers). These front end crates could be reserved for read/write or readonly (ie, can't be reset or intitialized). It was the responsibility of the run coordinator to schedule the detector so that users could get write access as needed. Partitions could come and go then, adding/removing trigger tables as necessary.

For example, say the run coordinator has made 4 highways available for the next two days. The pixel group could reserve the pixel front end electronics and associated L1 buffers for read/write, load the pixel trigger table. Crossing would be distributed to all 4 highways. Online software would assign specific L2/L3 nodes to this partition as constrained by the run coordinator. The silicon strip group could come and

reserve silicon electronics for read/write and pixel for read only and load a second set of trigger tables. Again, the software would assign L2/L3 worker nodes specifically to this partition. If a given crossing passed the L1 trigger for both partitions, it could be routed to worker nodes in both partitions or split between the two partitions in a predefined scalar. This was still also being discussed in the collaboration.

Partioning became an obvious solution when discussing the problem of how to utilize spare online cycles for offline. The pure Computer Scientists involved in RTeS promoted real time scheduling on the worker nodes to squeak out the maximum CPU utilization, but they were outweighed by the opinion that a particular worker node should be single tasking for a more simplistic and manageable from an operational standpoint.

Nodes could manually by moved between online and offline partitions, but we also envisioned an automatic shift toward offline as luminosity in the tevatron dropped. An overall luminosity profile would be loaded at the start of a run. As system monitoring tools detected lower utilization in the worker nodes, they would migrate nodes into the offline partition.

5. Conclusions

Thanks to the efforts of many talented people in the collaboration and at the lab as well as extremely helpful comments from the many reviews, the DAQ and trigger groups in BTeV were able to develop a highly credible online architecture. This architecture was believable in terms of cost and schedule with well understood, and mimimal risks.

Key elements in the success of the design were developing an architecture which maximized the number of commodity components (switching fabric, trigger farms) and on minimizing the variability between custom boards, i.e, the DCBs were common across all detectors.

6. References

[1- your url is the personnel pointer... You might consider instead
<http://www-btev.fnal.gov/public/hep/detector/rtes/>

http://www-btev.fnal.gov/cgi-bin/DocDB/ShowDocument?docid=4034
the (RTAS/SIGBED) document itself. I will provide you with
a "submitted for publication in ..." citation when DocDB 4034
is submitted (Friday this week, I hope), but if you look ***inside***
there is a really nice 2005 ARMOR citation (IEEE Internet Computing),
for example, which actually has a section on BTeV.

Article Title", *Journal*, Publisher, Location, Date, pp. 1-10.

[2] Jones, C.D., A.B. Smith, and E.F. Roberts, *Book Title*, Publisher, Location, Date.

(the above are dummies, to show format)

[3] http://www-btev.fnal.gov/public/hep/detector/rtes/index.shtml

[4] http://www-btev.fnal.gov/public/GeneralInformation.shtml

[5] http://www-btev.fnal.gov/cgi-bin/DocDB/ListBy?topicid=96

[6] http://www-btev.fnal.gov/public/hep/detector/rtes/Publications/index.html

[gme1] Karsai G., Sztipanovits J., Ledeczi A., Bapty T., "Model-Integrated Development of Embedded Software", Proceedings of the IEEE, Vol. 91, Number 1, pp. 145-164, January, 2003.

[gme2] Ledeczi A., Maroti M., Bakay A., Karsai G., Garrett J., Thomason IV C., Nordstrom G., Sprinkle J., Volgyesi P., "The Generic Modeling Environment", Workshop on Intelligent Signal Processing, Budapest, Hungary, May 17, 2001.

[make1] J. Sztipanovits, G. Karsai, "Model-Integrated Computing", IEEE Computer, pp. 110-112, April, 1997.

[make2] Universal Data Model tools available from ISIS and the Escher Research Institute http://escher.isis.vanderbilt.edu/tools/get_tool?UDM

[vla] Messie, D., "Polymorphic Self-* Agents for Stigmergic Fault Mitigation in Large-Scale Real-Time Embedded Systems", Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Utrecht, The Netherlands, July, 2005.

[1] Z. Kalbarczyk, R. K. Iyer, and L. Wang, "Application Fault Tolerance with Armor Middleware," *IEEE Internet Computing*, Special Issue on Recovery-Oriented Computing, March/April 2005.

[2] K. Whisnant, R. Iyer, Z. Kalbarczyk, et al. "The Effects of an ARMOR-Based SIFT

Environment on the Performance and Dependability of User Applications," in *IEEE Transactions on Software Engineering*, 30(4), 2004.

[3] 21. K. Whisnant, Z. Kalbarczyk, R. Iyer, "A System Model for Reconfigurable Software," in *IBM Systems Journal*, 42(1), 2003.

[4] Z. Kalbarczyk, et al. "Chameleon: A software infrastructure for adaptive fault tolerance," in *IEEE Trans. on Parallel and Distributed Systems*, 10(6), 1999.